Gibran Darmawan

13520061

Prodi Teknik Informatika

Sekolah Tinggi Informatika dan Elektro

1. Algoritma

Branch and Bound merupakan salah satu cara untuk menyelesaikan permasalahan yang menjadi materi dalam mata kuliah Strategi Algoritma. Algoritma Branch and Bound bekerja dengan mempertimbangkan sebuah himpunan kandidat-kandidat solusi. Algoritma ini menjelajahi *cabang* dari kandidat solusi. Kemudian cabang ini dibandingkan dengan batas solusi optimal dan apabila tidak ada solusi yang lebih baik, akan dibuang. Implementasi algoritma ini dalam puzzle 15 dengan mengeksplorasi pergerakan angka 16 (biasa disebut blank) dan menentukan sebuah ongkos (*cost*) paling kecil untuk himpunan solusinya. Program terus menerus menngimplementasikan yang disebelumnya sehingga menemukan solusi akhir.

2. Source Code

```
import copy
from queue import PriorityQueue as PQ
import numpy as np

patok = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
patokmatriks = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

def kurang(matriks):
    list = toList(matriks)
    ct,kurang=0,0
    isikurang = [0 for i in range(16)]
    for i in list:
        for j in range(ct, len(list)):
            if i == 16 and (list.index(i) %2) == 0:
                kurang += 1
            if i > list[j]:
                isikurang[i-1] += 1
                kurang += 1
        ct+=1

    for i in range(len(isikurang)):
        print(f"Kurang({i+1}) = {isikurang[i]}")


    print(f"Nilai = {kurang} + {list.index(16) %2}")
    print("kurang = ",kurang)
    return kurang

def cost(matriks):
    cost = 0
```

```
    list = toList(matriks)
    for i in range(len(list)):
        if list[i] != patok[i] and (list[i] != 16):
            cost += 1

    return cost

def toList(matrix):

    list = [0 for i in range(16)]
    cnt = 0
    for i in range(4):
        for j in range(4):
            list[cnt] += matrix[i][j]
            cnt+=1
    return list

def bematriks(list):
    matriks = [[0 for i in range(4)],[0 for i in range(4)],[0 for i in range(4)],[0 for i in range(4)]]
    for i in range(4):
        matriks[0][i]=list[i]
    for j in range(4,8):
        matriks[1][j-4]=list[j]
    for k in range(8,12):
        matriks[2][k-8]=list[k]
    for a in range(12,16):
        matriks[3][a-12]=list[a]
    return matriks



def getBlankIndex(matriks):
    for i in range(4):
        for j in range(4):
            if matriks[i][j] == 16:
                return [i, j]

def swapAtas(matriks):
    x,y = getBlankIndex(matriks)
    if x == 0:
        return matriks
    else:
        matriks[x-1][y], matriks[x][y] = matriks[x][y], matriks[x-1][y]

        return matriks

def swapKiri(matriks):
    x,y = getBlankIndex(matriks)
    if y == 0:
```

```
        return matriks
    else:
        matriks[x][y-1], matriks[x][y] = matriks[x][y], matriks[x][y-1]
        return matriks

def swapBawah(matriks):
    x,y = getBlankIndex(matriks)
    if x == 3:
        return matriks
    else:
        matriks[x+1][y], matriks[x][y] = matriks[x][y], matriks[x+1][y]
        return matriks

def swapKanan(matriks):
    x,y = getBlankIndex(matriks)
    if y == 3:
        return matriks
    else:
        matriks[x][y+1], matriks[x][y] = matriks[x][y], matriks[x][y+1]
        return matriks

def leastCost(matriks1, matriks2, matriks3, matriks4):
    apadah = [cost(matriks1), cost(matriks2), cost(matriks3),cost(matriks4)]
    return min(apadah), apadah.index(min(apadah))



class Node:
    def __init__(self, parent, matriks, cost, blank, level):
        #parent
        self.parent = parent
        #matriks
        self.matriks = matriks
        # fungsi cost
        self.cost = cost
        # posisi block kosong
        self.blank = blank
        # level nodenya
        self.level = level

    def __lt__(self, other):
        return(self.cost+self.level <= other.cost + other.level)

def mulai(matriks):
    visited = set()
    antrian = PQ()
    visited.add(tuple(np.reshape(matriks,16)))

    costroot = cost(matriks)
```

```
    blankroot = getBlankIndex(matriks)
    root = Node(None, matriks, costroot, blankroot, 0)
    antrian.put(root)
    start_time = time.time()


    while not antrian.empty():
        node = antrian.get()
        if node.cost == 0:
            endtime = time.time()
            print(f"Waktu = {endtime-start_time}")
            return node.matriks

        else:
            matriks1,matriks2,matriks3,matriks4 =
copy.deepcopy(node.matriks),copy.deepcopy(node.matriks),copy.deepcopy(node.matriks),copy.deepcopy(
node.matriks)
            nani= [swapAtas(matriks1),swapBawah(matriks2),swapKanan(matriks3),swapKiri(matriks4)]
            for matriksmana in nani:
                if tuple(np.reshape(matriksmana,16)) not in visited:
                    visited.add(tuple(np.reshape(matriksmana,16)))
                    child_cost = cost(matriksmana)
                    child_blank = getBlankIndex(matriksmana)
                    child = Node(node, matriksmana, child_cost, child_blank, node.level+1)
                    antrian.put(child)
                    print(child.matriks)
```

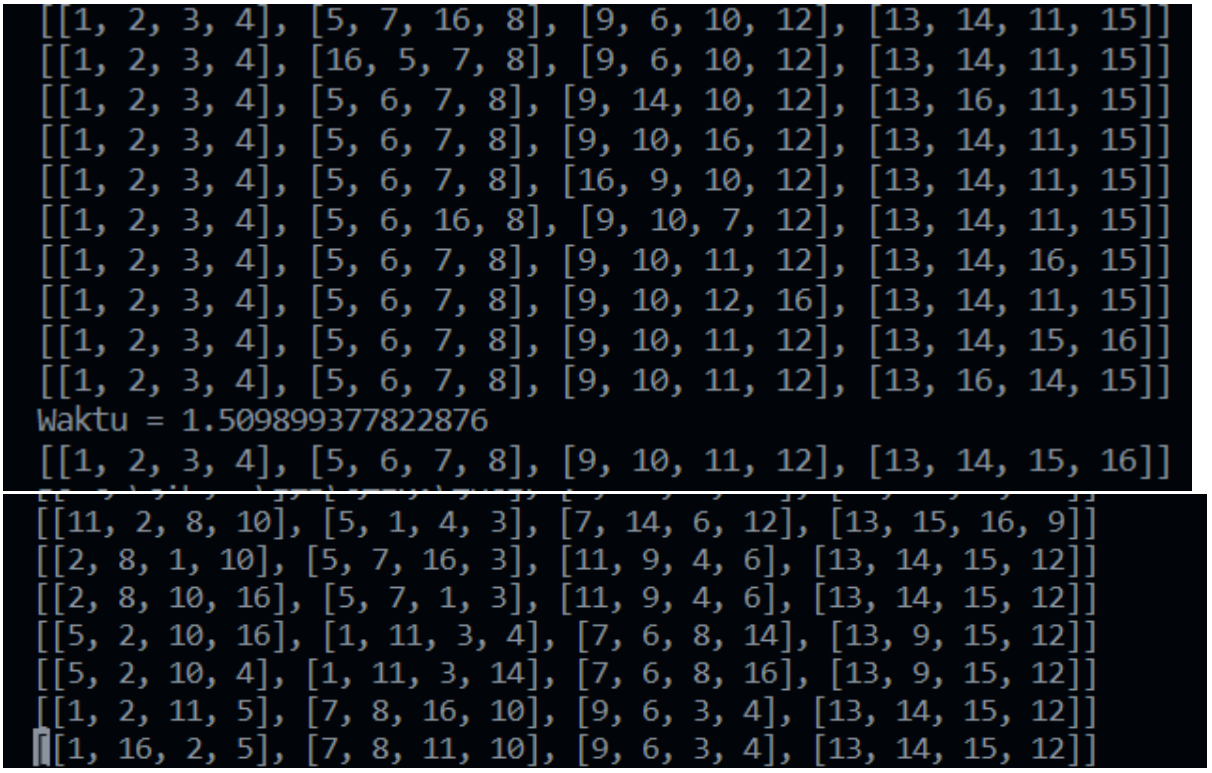
3. **Test Case**
   a.

```
[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,16,14,15]]
```

```
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 16, 11, 12], [13, 10, 14, 15]]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 16, 15]]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [16, 13, 14, 15]]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 16, 12], [13, 14, 11, 15]]
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
Waktu = 0.0018427371978759766
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
```

   b.

```
startPuzzle = [[1,16,3,4],[5,6,7,8],[9,10,11,12],[13,2,14,15]]
```

c. 

Program lama menjalankannya (program kurang efektif)

4. Link Github

https://github.com/gibrandarmawan/Tucil3-STIMA

5. Table Checklist

Poin	Ya	Tidak
1. Program berhasil dikompilasi	V	
2. Program berhasil running	V	
3. Program dapat menerima input dan menuliskan output.	V	
4. Luaran sudah benar untuk semua data uji		v
5. Bonus dibuat		v